



***¡LES DAMOS LA
BIENVENIDA!***

¿Están listos?

***RECUERDA PONER A GRABAR LA
CLASE***

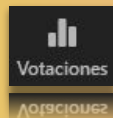


PRESENTACIÓN DE ESTUDIANTES



Por encuestas de Zoom:

1. País
2. Conocimientos previos en (temática del curso)
3. ¿Por qué elegiste el curso?



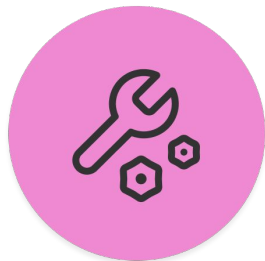


¿DUDAS DEL ON-BOARDING?

MIRALO AQUI

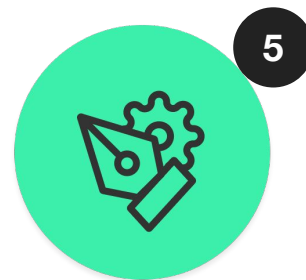
DESAFÍOS Y ENTREGABLES

Son actividades o ejercicios que se realizan durante la cursada, para enfocarse en la práctica.



Desafíos genéricos

Ayudan a poner en práctica los conceptos y la teoría vista en clase. No deben ser subidos a la plataforma.



Desafíos entregables

Relacionados completamente con el **Proyecto Final**. Deben ser subidos obligatoriamente a la plataforma hasta 7 días luego de la clase para que sean corregidos.

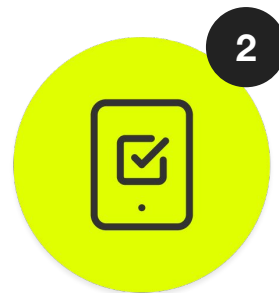
DESAFÍOS Y ENTREGABLES

Son actividades o ejercicios que se realizan durante la cursada, para enfocarse en la práctica.



Desafíos complementarios

Desafíos que complementan a los entregables. Son optativos y, de ser subidos a la plataforma a tiempo y aprobados, suman puntos para el top 10.



Entregas del Proyecto Final

Entregas con el estado de avance de tu **proyecto final** que deberás subir a la plataforma a lo largo del curso y hasta 7 días luego de la clase, para ser corregidas por tu docente o tutor/a.



PROYECTO FINAL

El Proyecto Final se construye a partir de los **desafíos** que se realizan clase a clase. Se va creando a medida que el estudiante sube los desafíos entregables a nuestra plataforma.

El objetivo es que cada estudiante pueda utilizar su Proyecto Final como parte de su portfolio personal.

El **proyecto final** se debe subir a la plataforma la ante-última o última clase del curso. *En caso de no hacerlo tendrás 20 días a partir de la finalización del curso para cargarlo en la plataforma. Pasados esos días el botón de entrega se inhabilitará.*

¿CUÁL ES NUESTRO PROYECTO FINAL?

Web Playground

Proyecto
Final



Desarrollarás una **web** en **Django**. Las entregas son individuales. Se verán temas avanzados:

Registros:

- Login/signup
- Reset pwd
- Logout

Páginas:

- CRUD sólo si está registrado

Perfil:

- Imagen
- Editar email/pwd

Admin:

- Apps



¿Que es una Web Playground?

Un playground **es un lugar de pruebas**, donde podremos hacer, deshacer, refactorizar, romper, etc.

En un Playground **no se abordan las cuestiones relativas a la Interfaz Gráfica (la parte visual de la App), sólo trabajaremos con el Código Fuente.**

La idea de generar una web playground es para que podamos aventurarnos en las funciones más avanzadas de Python y Django, de tal forma que siempre tengamos un lugar donde probar dichas funciones.

Los desafíos entregables y el Proyecto Final se realizarán en EQUIPOS



- Fijos para toda la cursada
- Conformados por **dos o tres** estudiantes
- Creados automáticamente y al azar por la plataforma de Coderhouse
- Cada equipo tendrá un mismo tutor/a
- Un integrante del equipo (puede ser cualquiera) sube la entrega a través de la plataforma y se verá reflejada en la plataforma de todo el equipo
- Cuando su tutor/a corrige la entrega, todo el equipo recibe la devolución
- Todas las calificaciones serán grupales



<i>ENTREGA</i>	<i>REQUISITO</i>	<i>FECHA</i>
1° entrega	Web Playground	Clase N° 21
Proyecto Final	Web Playground Advanced	Clase N° 25

¡IMPORTANTE!

Los desafíos y entregas de proyecto se deben cargar hasta siete días después de finalizada la clase. Te sugerimos llevarlos al día.

LUNES 16/03 20:30HS

🕒

10. Estrategia de contenido para Twitter y LinkedIn

👍

● HOY 20:30

📁

Tenes tiempo hasta el
23/03/2020

↑ ENTREGAR

DESAFÍO - EXPIRA EL 23/03/2020

Crear publicaciones para Twitter



Clase 1. Python

Números y cadenas de caracteres



OBJETIVOS DE LA CLASE

- Implementar operaciones básicas y avanzadas de números
- Asignar variables
- Implementar operaciones numéricas con variables
- Operar con cadena de caracteres
- Reconocer funcionalidades de cadenas de caracteres

CRONOGRAMA DEL CURSO

Clase 0



Introducción a programar con Python



KICK OFF



INSTALACIÓN DE PYTHON Y COLABS

Clase 1



Números y cadenas de caracteres



DESAFÍO DE NÚMEROS



DESAFÍO DE STRING



DESAFÍO DE SLICING



MI PRIMER PROGRAMA

Clase 2



Listas y tuplas



DESAFÍO DE LISTAS



DESAFÍO DE TUPLAS



PRIMERAS PRÁCTICAS

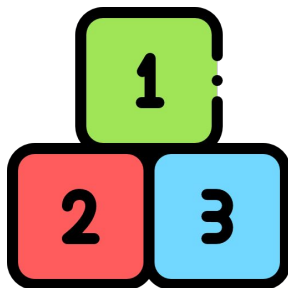
NÚMEROS

TIPOS DE NÚMERO

Números en python

Los números de Python están relacionados con los números matemáticos, pero **están sujetos a las limitaciones de la representación numérica en las computadoras.**

Python distingue entre **enteros**, números de **punto flotante** y números **complejos**.





Enteros

Los números enteros son aquellos que **no tienen decimales**, tanto positivos como negativos (además del cero). En Python se pueden representar mediante el **tipo `int`** (de integer, entero) o el **tipo `long`** (largo).

La única diferencia es que el tipo **`long` permite almacenar números más grandes**. Es aconsejable no utilizar el tipo `long` a menos que sea necesario, para no malgastar memoria.

Ejemplo: 1, 2, 525, 0, -817



Long

Los números enteros **largos** o **long** en python son iguales a los enteros, no tienen decimales, y pueden ser positivos, negativos o cero. Se tratan de números de cualquier tamaño. Se puede definir con una **L** al costado de nuestro número.

Ejemplo:

```
9812893712387912379123L
```

```
897538475389475198237891249823L
```

```
12387349587373L
```



Float / Decimal

Los números reales, son los que tienen decimales, en python se expresan mediante el tipo **float**. Desde Python 2.4 cuenta con un nuevo tipo Decimal, para el caso de que se necesite representar fracciones de forma más precisa.

Ejemplo:

0,270

-12,1233

989,87439124387

-74,9349834

CODER HOUSE



Complejos

Los números complejos son los que tienen parte imaginaria, es muy probable que no lo vayas a necesitar nunca. Este tipo se llama **complex**, se almacena usando reales ya que es una extensión de dichos números.

Ejemplo:

$2,1j$

$-41,832i$

$88,23\ 254j$

OPERACIONES NUMÉRICAS



Operaciones numéricas en Python

En programación y en matemáticas, los **operadores aritméticos** son aquellos que manipulan los datos de tipo numérico, es decir, **permiten la realización de operaciones matemáticas** (sumas, restas, multiplicaciones, etc).

El resultado de una operación aritmética es un dato aritmético, es decir, si ambos valores son números enteros el resultado será de tipo entero; si alguno de ellos o ambos son números con decimales, el resultado también lo será.



Operadores aritméticos en python

Operación	Operador	Ejemplo
Suma	+	$3+5 = 8$
Resta	-	$4 - 1 = 3$
Multiplicación	*	$3 * 6 = 18$
Potencia	**	$3 ** 2 = 9$
División (cociente)	/	$15.0 / 2.0 = 7.5$
División (parte entera)	//	$15.0 // 2.0 = 7$
División (resto)	%	$7 \% 5 = 1$

PROCEDENCIA



Precedencia de los operadores

Al igual que ocurre en matemáticas, en programación también tenemos una **prioridad en los operadores**. Esto significa que si una expresión matemática es precedida por un operador y seguido de otro, el operador más alto en la lista debe ser aplicado por primera vez.

Las expresiones con paréntesis se evalúan de dentro a fuera, el paréntesis más interno se evalúa primero.



Precedencia de los operadores

El orden normal de las operaciones es de izquierda a derecha, evaluando en orden los siguientes operadores:

1. Términos entre paréntesis.
2. Potenciación y raíces.
3. Multiplicación y división.
4. Suma y resta.



Precedencia de los operadores

En el lenguaje de programación de Python se representan los operadores con el siguiente orden:

1. `()`
2. `**`
3. `X, /, %, //`
4. `+, -`

CADENAS DE TEXTO



Cadenas de texto en python

Las cadenas (o strings) son un tipo de **datos compuestos por secuencias de caracteres que representan texto.**

Estas cadenas de texto son de **tipo str** y se delimitan mediante el uso de comillas simples o dobles.

Ejemplo:

“Esto es una cadena de texto”

‘Esto también es una cadena de texto’

CODER HOUSE



Cadenas de texto en python

En el caso que queramos usar comillas (o un apóstrofe) dentro de una cadena tenemos distintas opciones. La más simple es encerrar nuestra cadena mediante un tipo de comillas (simples o dobles) y usar el otro tipo dentro de la cadena.

AaI

Cadenas de texto en python

Otra opción es **usar** en todo momento **el mismo tipo de comillas**, pero **usando** **la barra invertida (\) como carácter de escape** en las comillas del interior de la cadena para indicar que esos caracteres forman parte de la cadena.

Ejemplos:

“Esto es un ‘texto’ entre comillas dobles”

‘Esto es otro “texto” entre comillas simples’

“Esto es otro \“texto\” todo en comillas dobles”

‘Esto otro \'texto\' todo en comillas simples’

CODER HOUSE

PRINT



¿Para qué sirve?

La forma correcta de mostrar cadenas de texto (u otros objetos) por pantalla en python es utilizando una función llamada **print** (imprimir). **Se indica lo que se desea mostrar por pantalla entre paréntesis.**

Ejemplo:

```
print("Esto es una cadena de texto")  
print("Esto también es una cadena de texto")  
print(4)
```



Ventajas

Usar print tiene sus ventajas. Por ejemplo, nos deja mostrar por pantalla caracteres especiales, como tabulación o saltos de línea.

Ejemplo:

```
[in]print('Una cadena\tcon tabulación')
```

```
[out]Una cadena    con tabulación
```

```
[in]print("Otra cadena\ncon salto de línea")
```

```
[out]Una cadena  
con salto de línea
```



Print

Si por ejemplo quisiéramos imprimir el directorio de una carpeta, sería de la siguiente forma:

```
print('C:\nombre\directorio')
```

Pero va a tomar el **\n** como carácter especial para salto de línea. Para poder ignorar estos caracteres especiales Python tiene una forma de “*printear*” cruda o raw.



Print

Lo indicamos con una **r** delante de lo que se va a imprimir, y Python automáticamente lo interpretará para no tomar en cuenta los caracteres especiales.

```
print(r'C:\nombre\directorio')
```



Print

Otra funcionalidad que tiene es permitir mostrar **una cadena en distintas líneas**, de forma que **con un sólo print se muestran varias líneas de cadenas**.

Para lograrlo tenemos que pasarlo entre **tres** comillas dobles, o **tres** comillas simples.

Ejemplo:

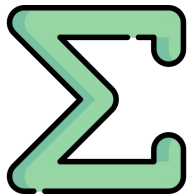
```
print("""una cadena  
otra cadena  
otra cadena más  
""")
```




BREAK

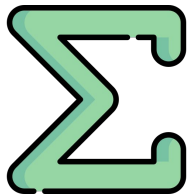
¡5/10 MINUTOS Y VOLVEMOS!

VARIABLES



Variables en Matemáticas

El concepto de "**variable**" proviene de las matemáticas, donde una variable es un símbolo que forma parte de una expresión o de una fórmula. Normalmente las variables se representan mediante letras del alfabeto latino (x, y, z, n, i, j, etc.).



Variables en Matemáticas

Dependiendo del contexto, las variables significan cosas distintas. En el caso del Álgebra, una variable representa una cantidad desconocida que se relaciona con otras. Consideremos por ejemplo la ecuación:

$$x + 3 = 5$$

VARIABLES EN PROGRAMACIÓN

Variables en programación

En algunos lenguajes de programación, las variables se pueden entender como "cajas" en las que se guardan los datos, pero **en Python las variables son "etiquetas" que permiten hacer referencia a los datos (que se guardan en unas "cajas" llamadas objetos).**

Python es un lenguaje de programación orientado a objetos y su modelo de datos también está basado en objetos.

Variables en programación

Para cada dato que aparece en un programa, Python crea un objeto que lo contiene. Cada objeto tiene:

- Un **identificador único** (un número entero, distinto para cada objeto). El identificador permite a Python referirse al objeto sin ambigüedades.
- Un **tipo de datos** (entero, decimal, cadena de caracteres, etc.). El tipo de datos permite saber a Python qué operaciones pueden hacerse con el dato.
- Un **valor** (el propio dato).

Variables en programación

```
31 def __init__(self, request):
32     self.file = None
33     self.fingerprints = set()
34     self.logdups = True
35     self.debug = debug
36     self.logger = logging.getLogger(__name__)
37     if path:
38         self.file = open(os.path.join(path, "requests.log"),
39                           "a")
40         self.file.seek(0)
41         self.fingerprints.update(self.request.headers)
42
43 @classmethod
44 def from_settings(cls, settings):
45     debug = settings.getbool("SUPERHERO_DEBUG")
46     return cls(job_dir(settings), debug)
47
48 def request_seen(self, request):
49     fp = self.request_fingerprint(request)
50     if fp in self.fingerprints:
51         return True
52     self.fingerprints.add(fp)
53     if self.file:
54         self.file.write(fp + os.linesep)
55
56 def request_fingerprint(self, request):
57     return request_fingerprint(request)
```

Las variables en Python no guardan los datos, sino que son simples nombres para poder hacer referencia a esos objetos.

Variables en programación



En Python, si escribimos la instrucción: **a = 2**

- Se crea el objeto "2". Ese objeto tendrá un identificador único que se asigna en el momento de la creación y se conserva a lo largo del programa. En este caso, el objeto creado será de tipo número **entero** y guardará el **valor** 2.
- Se asocia el nombre a al objeto número entero 2 creado.



Variables en programación



Al describir la instrucción anterior no habría que decir 'la variable **a** almacena el número entero **2**', sino que habría que decir 'podemos llamar **a** al objeto número entero **2**'. **La variable a es como una etiqueta que nos permite hacer referencia al objeto "2"**, más cómoda de recordar y utilizar que el identificador del objeto.



DEFINIR LAS VARIABLES



Definir una variable



Las variables en Python se crean cuando se definen por primera vez, es decir, **cuando se les asigna un valor por primera vez.**

Para asignar un valor a una variable se utiliza el operador de igualdad **(=)**. A la izquierda de la igualdad se escribe el nombre de la variable y a la derecha el valor que se quiere dar a la variable.

```
>>> mi_variable = 2
```



Definir una variable



Siempre se escribe a la izquierda de la igualdad, de lo contrario, Python generará un mensaje de error:

```
>>> 2 = mi_variable  
SyntaxError: can't assign to literal
```

Para mostrar el valor de la variable hay que escribir su nombre, o “printearlo”.

```
>>> mi_variable = 2  
>>> mi_variable  
2  
>>> print(mi_variable)  
2
```



Definir una variable



Si una variable no se ha definido previamente, al escribir su nombre o printear la variable generará un error:

```
>>> x = -10
```

```
>>> y
```

Traceback (most recent call last):

File "<pyshell#1>", line 1, in <module>

y

NameError: name 'y' is not defined



Definir una variable



Una variable puede almacenar números, texto o estructuras más complicadas (que se verán más adelante). **Si se va a almacenar texto, el texto debe escribirse entre comillas simples (') o dobles ("), que son equivalentes.** A las variables que almacenan texto se les suele llamar cadenas (de texto).

```
>>> nombre = "Pepito Conejo"
```

```
>>> nombre
```

```
'Pepito Conejo'
```

```
>>> print(nombre)
```

```
'Pepito Conejo'
```



Definir una variable



Si no se escriben comillas, Python supone que estamos haciendo referencia a otra variable (que, si no está definida, genera un mensaje de error):

```
>>> nombre = Pepe
```

```
Traceback (most recent call last):
```

```
File "<pyshell#0>", line 1, in <module>
```

```
    nombre = Pepe
```

```
NameError: name 'Pepe' is not defined
```

```
>>> nombre = Pepito Conejo
```

```
SyntaxError: invalid syntax
```




Aunque no es obligatorio, **se recomienda que el nombre de la variable esté relacionado con la información que se almacena en ella para que sea más fácil entender el programa.**



Si el programa es trivial o mientras se está escribiendo un programa, esto no parece muy importante, pero si se consulta un programa escrito por otra persona o escrito por uno mismo hace tiempo, resultará mucho más fácil entender el programa si los nombres están bien elegidos.



Nombres de variables

El nombre de una variable debe empezar por una letra o por un guión bajo (_) y puede seguir con más letras, números o guiones bajos (esto en ingles se llama **snake case**).

```
>>> fecha_de_nacimiento = "27 de octubre de 1997"
```

```
>>> fecha_de_nacimiento
```

```
'27 de octubre de 1997'
```

Los nombres de variables no pueden incluir espacios en blanco.

```
>>> fecha de nacimiento = "27 de octubre de 1997"
```

SyntaxError: invalid syntax

CODER HOUSE



Nombres de variables

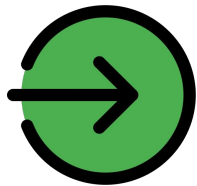
Los nombres de las variables pueden contener mayúsculas, pero tenga en cuenta que **Python distingue entre mayúsculas y minúsculas** (en inglés se dice que Python es *case-sensitive*).



Nombres de variables

```
>>> nombre = "Pepito Conejo"  
>>> Nombre = "Numa Nigerio"  
>>> nomBre = "Fulanito Mengáñez"  
>>> nombre  
'Pepito Conejo'  
>>> Nombre  
'Numa Nigerio'  
>>> nomBre  
'Fulanito Mengáñez'
```

INPUT



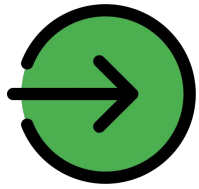
Input



En Informática, la "**entrada**" o **input** de un programa son los datos que llegan al programa desde el exterior. Actualmente, el origen más habitual es el teclado.

Python tiene una función llamada **input()** la cual **permite obtener texto escrito por teclado**. Al llegar a la función, el programa se detiene esperando que se escriba algo y se pulse la tecla **Intro**.

```
Ej: >>> nombre = input()
```



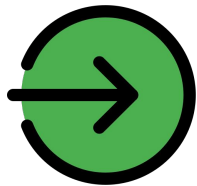
Input



Otra solución, más compacta, es aprovechar que a la función **input()** se le puede enviar un argumento que se escribe en la pantalla (sin añadir un salto de línea):

Ej:

```
>>> nombre = input("¿Cómo te llamas?")
```

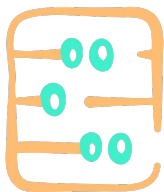
Conversión de tipos

De forma predeterminada, la función `input()` convierte la entrada en una cadena, aunque escribamos un número. Si intentamos hacer operaciones, se producirá un error.

Si se quiere que Python intérprete la entrada como un número entero, se debe utilizar la función `int()` de la siguiente manera:

```
Ej: >>> nombre = int(input("¿Que edad tenes?"))
```

OPERACIONES ARITMÉTICAS CON VARIABLES



Operaciones aritméticas con variables

Podemos utilizar todos los operadores aritméticos antes vistos en las variables numéricas. Algunos ejemplos:

```
>>> a = 2
```

```
>>> b = 3
```

```
>>> a+b
```

5

```
>>> a = 5
```

```
>>> b = 2
```

```
>>> a * b
```

10

```
>>> a = 35
```

```
>>> b = 7
```

```
>>> a / b
```

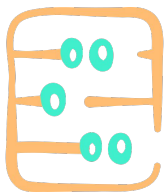
5

```
>>> a = 3
```

```
>>> b = 2
```

```
>>> a ** b
```

9



Operaciones aritméticas con variables

Podemos utilizar todos los operadores aritméticos antes vistos en las variables de string también. Algunos ejemplos:

```
>>> cadena = "Python"  
>>>cadena * 2  
"PythonPython"
```

```
>>>cadena = "Python"  
>>>otra_cadena = "Hola!"  
>>>otra_cadena + cadena  
"Hola!Python"
```

A la suma de cadenas de caracteres la llamaremos **concatenación**

CODER HOUSE

INDEXACIÓN DE STRINGS

Indexación de las cadenas de texto

Cada uno de los caracteres de una cadena (incluidos los espacios) tiene asignado un índice. Este índice nos permite seleccionar su carácter asociado haciendo referencia a él entre corchetes ([]) en el nombre de la variable que almacena la cadena.



Indexación de las cadenas de texto

Si consideremos el orden de izquierda a derecha, el índice empieza en 0 para el primer carácter, etc.

También se puede considerar el orden de derecha a izquierda, en cuyo caso al último carácter le corresponde el índice -1, al penúltimo -2 y así sucesivamente.





Indexación de las cadenas de texto



Este método es útil si por ejemplo queremos acceder a caracteres en las últimas posiciones de una cadena con muchos caracteres de la cual no conocemos su longitud.

```
>>> cadena = "Python"
```

```
>>> cadena[0]
```

```
'P'
```

```
>>> cadena[-1]
```

```
'n'
```

Caracteres :	P	y	t	h	o	n
Índice :	0	1	2	3	4	5
Índice inverso :	-6	-5	-4	-3	-2	-1

CODER HOUSE

LONGITUD DE STRINGS

Longitud de string

Python nos da una función llamada **len**. Esta función nos **permite saber cuál es la longitud de un string**, sin la necesidad de contar uno a uno los caracteres que tiene. También nos sirve en el caso de que no sepamos qué valor tiene una variable, pero tenemos que sacar determinados caracteres por índice.

Longitud de string

Ejemplo de len:

```
>>> palabra = "Python"
```

```
>>> len(palabra)
```

```
6
```

```
>>> otra_palabra = "Hola, como están? Yo bien!"
```

```
>>> len(otra_palabra)
```

```
26
```

SLICING

Rebanar string (slicing)

Otra función de las cadenas que podemos usar, es seleccionar solamente una parte de las cadenas. Para ello se usa la notación `[inicio:fin:paso]` también en el nombre de la variable que almacena la cadena

- **Inicio:** es el índice del primer carácter de la porción de la cadena que queremos seleccionar. 🙌

Rebanar string (slicing)

- **Fin:** es el índice del último carácter no incluido de la porción de la cadena que queremos seleccionar.
- **Paso:** indica cada cuantos caracteres seleccionamos entre las posiciones de inicio y fin.

```
>>> cadena = "Python"
```

```
>>> cadena[0:4:1]
```

```
'Pyth'
```

```
>>> cadena[2:6:2]
```

```
'to'
```

¿Reasignar valor?

En algún momento nos preguntaremos si es posible traer el valor de una cadena de un índice a otro, ¿eso significa que puedo cambiarle el valor de un índice a uno que yo quiera?

Observemos el siguiente ejemplo:

```
>>> palabra = "Python"
```

¿Reasignar valor?

¡Cometimos un error! Debería decir Python, no Pithon

```
>>> palabra[1] = "y"
```

Traceback (most recent call last):

File "<pyshell#0>", line 1, in <module>

```
    palabra[1] = "y"
```

TypeError: 'str' object does not support item assignment

¿Reasignar valor?

Pero, ¿qué pasó acá? ¿Por qué se rompió?





En python, las cadenas de texto o strings, son **INMUTABLES** esto significa, que no se puede sustituir ninguno de sus caracteres individualmente.

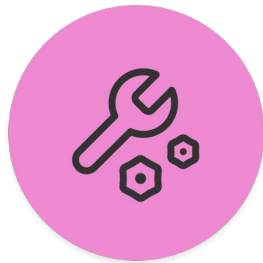
Pero esto no es un gran problema. 😎

Python es flexible, ¡podemos modificar el string que deseemos con slicing!

```
>>> palabra = "Pithon"
```

```
>>> palabra = palabra[0:1] + "y" + palabra[2:]
```

De esta forma podremos mostrar Python y no Pithon.



Desafío Números

Crear un programa para calcular la nota final de un estudiante

Tiempo estimado: 10 minutos



Nota Final

Crear un programa para calcular la nota final de un estudiante en base a dos exámenes, los exámenes cuentan con un porcentaje distinto de la nota final

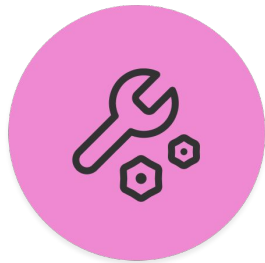
- **nota_1** cuenta como el 40% de la nota final
- **nota_2** cuenta como el 60% de la nota final

Nos organizaremos en Break Out rooms de 3 integrantes para realizar la actividad, con el tutor en común como facilitador.



Esto se suele llamar un **promedio pesado o ponderado**, donde **no todos los valores tienen el mismo “peso” o valor.**

1. **Promedio entre 3 y 10 es :** $(1.3 + 1.10) / 2$, este es el promedio tradicional donde todos los valores tienen un peso de 1.
2. **Promedio pesado entre 3 y 10 es:** $(13.3 + 2.10) / 15$, acá vemos que el peso de 3 es 13, y el peso del 10 es 2, por lo que el 3 es más importante, se divide por la suma de los pesos.



Desafío String

Genera una nueva variable

Tiempo estimado: 10 minutos



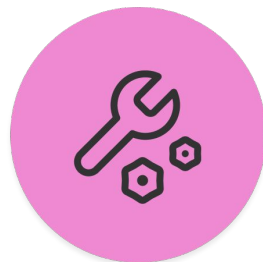
Reordenación

Dadas cuatro variables con diferentes textos (de forma individual), genera una nueva variable con el siguiente contenido:

Objetivo: “Python es un lenguaje de programación moderno”

Partiendo de:

1. `cadena_1 = “moderno”`
2. `cadena_2 = “Python”`
3. `cadena_3 = “es un lenguaje”`
4. `cadena_4 = “de programación”`



Desafío Slicing

Dar vuelta la cadena y asignarla a una variable

Tiempo estimado: 10 minutos



FORMATEAR

Se tiene una cadena de texto, pero al revés. Al parecer contiene el nombre de un alumno, la nota de un examen y la materia.

De forma individual, realiza lo siguiente:

1. Dar vuelta la cadena y asignarla a una variable llamada `cadena_volteada`. Para devolver una cadena dada vuelta se usa el tercer índice negativo con slicing: `cadena[::-1]`.
2. Extraer nombre y apellido, almacenarlo en una variable llamada `nombre_alumno`



FORMATEAR

4. Extraer la nota y almacenarla en una variable llamada nota.
5. Extraer la materia y almacenarla en una variable llamada materia.
6. Mostrar por pantalla la siguiente estructura:

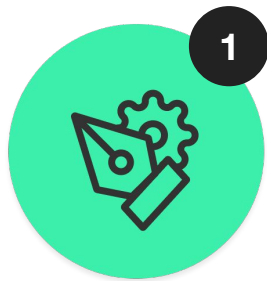
🕶️ *NOMBRE APELLIDO* ha sacado un *NOTA* en *MATERIA*

Formateando las anteriores variables en una variable llamada
cadena_formateada

cadena = "acitametaM ,5.8 ,otipeP ordeP"

👉 *Para formatear ¡recuerda concatenar!*





MI PRIMER PROGRAMA EN PYTHON

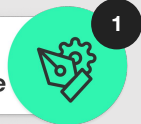
Crear un programa para calcular la nota final de los estudiantes.

MI PRIMER PROGRAMA EN PYTHON

Formato: El documento debe presentarse en Google Docs o Colabs bajo el siguiente formato: `"MiPrimerProgramaEnPython+Apellido"`.

Sugerencia: En el formulario debe estar el print de pantalla de la consola de Python con el ejercicio resuelto, como así también el código tipeado.

Desafío
entregable



>> Consigna: Crear un programa para calcular la nota final del estudiante en base a dos exámenes, los exámenes cuentan con un porcentaje distinto de la nota final

- **nota_1** cuenta como el 40% de la nota final
- **nota_2** cuenta como el 60% de la nota final

>>Aspectos a incluir en el entregable:

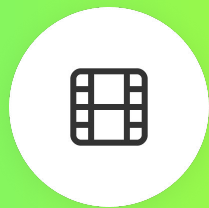
Tener en cuenta los temas vistos en la clase 1: números, print, input, variables, operaciones matemáticas, cadena de texto.

>>Aspectos a tener en cuenta:

Los datos deben guardarse en variables y deben ser dinámicos por medio de input.

¿PREGUNTAS?





***¿QUIERES SABER MÁS? TE DEJAMOS
MATERIAL AMPLIADO DE LA CLASE***




- Artículo: [Tipo Número](#)
- Artículo: [Operadores aritméticos](#)
- Artículo: [Tipo string](#)
- Artículo: [Variables](#)
- Artículo: [Print](#)



***¿AÚN TE QUEDASTE CON GANAS DE MÁS?
TE RECOMENDAMOS EL SIGUIENTE
MATERIAL***

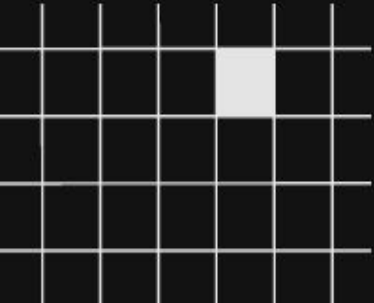


- [Variables Numéricas](#) | **Nicolás Perez**
- [Operaciones](#) | **Nicolás Perez**
- [Variables de Texto](#) | **Nicolás Perez**
- [EjemplosClase](#) 



¡MUCHAS GRACIAS!

Resumen de lo visto en clase hoy:

- Números
 - Strings
 - Print
 - Variables
 - Index & Slicing
- 



OPINA Y VALORA ESTA CLASE

#DEMOCRATIZANDO LA EDUCACIÓN

CODER HOUSE